CHAPTER 4
Bootstrap JavaScript Plugins

The components discussed in Chapter 3 are just the beginning. Bootstrap comes bundled with 13 jQuery plugins that extend the features and can add more interaction to your site. To get started with Bootstrap's JavaScript plugins, you don't need to be an advanced JavaScript developer. In fact, by utilizing Bootstrap Data API, most of the plugins can be triggered without writing a single line of code.

Overview

Bootstrap can be included on your site in two forms, either compiled or raw. In Bootstrap 2.2.2, the uncompressed file is 59KB and the minimized version is only 32KB. All of the Bootstrap plugins are accessible using the included Data API. With this, you don't need to include a single line of JavaScript to invoke any of the plugin features.

Typically, JavaScript lies in a either a separate file or at the bottom of the page before the closing `</body>` tag. You can either use the `src` attribute to link to another file, or you can write the contents of the file between the opening and closing tags:

```
<!-- To reference another JavaScript file -->
<script src="assets/js/javaScript.js"></script>

<!-- To write JavaScript to the page -->
<script type="text/javaScript">
        function js_alert{
                alert('Page has loaded');
        }
</script>
```

Generally, it is best to include all JavaScript calls into a check that ensures that the DOM has been loaded on the page. If you have the JavaScript trying to fire earlier, it may miss elements as the browser parses the page. With jQuery, adding a check is easily done by

selecting the `document` or the entire content of the page, and then applying the `.ready()` method:

```
$(document).ready(function(){
        alert('Page has loaded');
        // Once the page has loaded and is ready, an alert will fire.
});
```

As mentioned above, Bootstrap has a Data API where you can write data attributes into the HTML of the page. If you need to turn off the Data API, you can unbind the attributes by adding the following line of JavaScript:

```
$('body').off('.data-api')
```

If you need to disable a single plugin, you can do it programmatically using the namespace of the plugin along with the `data-api` namespace:

```
$('body').off('.alert.data-api')
```

Programmatic API

The developers of Bootstrap believe that you should be able to use all of the plugins throughout the JavaScript API. All public APIs are single, chainable methods and return the collection acted upon.

```
$('.btn.danger').button('toggle').addClass('active')
```

All methods should accept an optional options object, a string which targets a particular method, or nothing (which initiates a plugin with default behavior).

```
$("#myModal").modal()   // initialized with defaults
$("#myModal").modal({ keyboard: false })       // initialized with no keyboard
$("#myModal").modal('show')       // initializes and invokes show immediately
```

Transitions

The transition plugin provides simple transition effects. A few examples include:

- Sliding or fading in modals
- Fading out tabs
- Fading out alerts
- Sliding carousel panes

Modal

A modal is a child window that is layered over its parent window (see Figure 4-1). Typically, the purpose is to display content from a separate source that can have some

interaction without leaving the parent window. Child windows can provide information, interaction, or more. I use them as a window for holding slideshows and login/registration information. The modal plugin is probably one of my favorite Bootstrap features.

To create a static modal window, use this code:

```
<div class="modal hide fade">
  <div class="modal-header">
    <button type="button" class="close" data-dismiss="modal" aria-hidden="true">
    &times;</button>
    <h3>Modal header</h3>
  </div>
  <div class="modal-body">
    <p>One fine body…</p>
  </div>
  <div class="modal-footer">
    <a href="#" class="btn">Close</a>
    <a href="#" class="btn btn-primary">Save changes</a>
  </div>
</div>
```

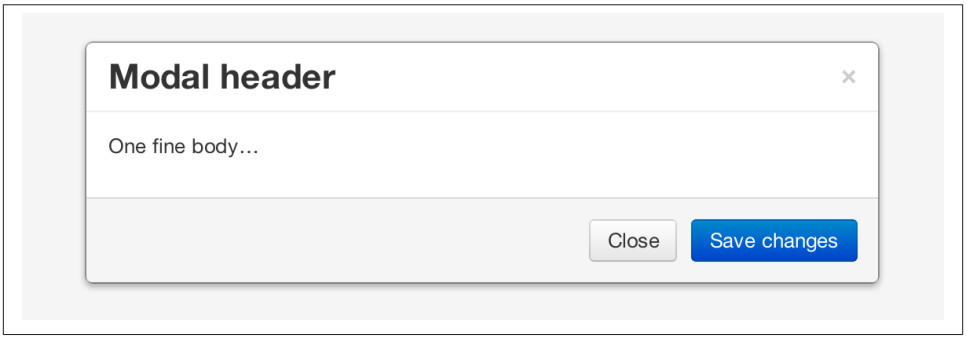

Figure 4-1. Static modal window

To invoke the modal window, you need to have some kind of a trigger. Normally I use a button or a link. If you look in the code below, you will see that in the `<a>` tag, the `href="myModal"` is the target of the modal that you want to load on the page. This code allows you to create multiple modals on the page and then have different triggers for each of them. Now, to be clear, you don't load multiple modals at the same time, but you can create many on the page to be loaded at different times.

There are three classes to take note of in the modal. The first is `.modal`, which is simply identifying the content of the `<div>` as a modal. The second is `.hide`, which tells the browser to hide the content of the `<div>` until we are ready to invoke it. And last, the `.fade` class. When the modal is toggled, it will cause the content to fade in and out.

```
<!-- Button to trigger modal -->
<a href="#myModal" role="button" class="btn" data-toggle="modal">Launch demo
modal</a>

<!-- Modal -->
<div id="myModal" class="modal hide fade" tabindex="-1" role="dialog"
aria-labelledby="myModalLabel" aria-hidden="true">
  <div class="modal-header">
    <button type="button" class="close" data-dismiss="modal"
    aria-hidden="true">×</button>
    <h3 id="myModalLabel">Modal header</h3>
  </div>
  <div class="modal-body">
    <p>One fine body…</p>
  </div>
  <div class="modal-footer">
    <button class="btn" data-dismiss="modal"
    aria-hidden="true">Close</button>
    <button class="btn btn-primary">Save changes</button>
  </div>
</div>
```

Usage

Using the Bootstrap JavaScript Data API, you simply need to pass a few data attributes to toggle the slideshow. To start with, set `data-toggle="modal"` on the link or button that you want to use to invoke the modal and then set the `data-target="#foo"` to the ID of the modal that you'd like to use.

To call a modal with `id="myModal"`, use a single line of JavaScript:

```
$('#myModal').modal(options)
```

Options

Options can either be passed in via data attributes or with JavaScript. To use the data attributes, prepend `data-` to the option name (e.g., `data-backdrop=""`). See Table 4-1 for descriptions of some modal options.

Table 4-1. Modal options

Name	Type	Default	Description
backdrop	Boolean	true	Set to false if you don't want the modal to be closed when the user clicks outside of the modal.
keyboard	Boolean	true	Closes the modal when escape key is pressed; set to false to disable.
show	Boolean	true	Shows the modal when initialized.
remote	path	false	Using the jQuery `.load` method, inject content into the modal body. If an `href` with a valid URL is added, it will load that content.

Methods

The following are some useful methods to use with modals.

Options

Activates your content as a modal. Accepts an optional options object.

`.modal(options)`.

```
$('#myModal').modal({
        keyboard: false
})
```

Toggle

Manually toggles a modal.

`.modal('toggle')`.

```
$('#myModal').modal('toggle')
```

Show

Manually opens a modal.

`.modal('show')`.

```
$('#myModal').modal('show')
```

Hide

Manually hides a modal.

`.modal('hide')`.

```
$('#myModal').modal('hide')
```

Events

Bootstrap provides the events listed in Table 4-2 if you need to hook into the function.

Table 4-2. Modal events

Event	Description
show	Fired after the `show` method is called.
shown	Fired when the modal has been made visible to the user.
hide	Fired when the `hide` instance method has been called.
hidden	Fired when the modal has finished being hidden from the user.

As an example, after the modal is hidden, you could cause an alert to fire:

```
$('#myModal').on('hidden', function () {
        alert('Hey girl, I heard you like modals...');
})
```

Dropdown

The dropdown was covered extensively in Chapter 3, but the interaction was glossed over. As a refresher, dropdowns can be added to the navbar, pills, tabs, and buttons.

Usage

To use a dropdown (Figure 4-2), add `data-toggle="dropdown"` to a link or button to toggle the dropdown.

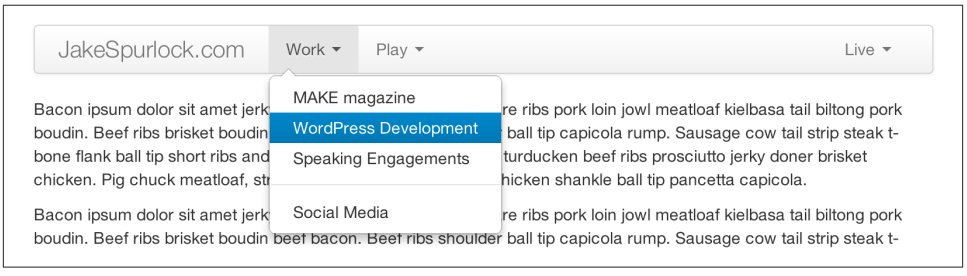

Figure 4-2. Dropdown within navbar

Here's the code for developing a dropdown with data attributes:

```
<li class="dropdown">
  <a href="#" id="drop" role="button" class="dropdown-toggle"
  data-toggle="dropdown">Word <b class="caret"></b></a>
  <ul class="dropdown-menu" role="menu" aria-labelledby="drop">
    <li><a tabindex="-1" href="#">MAKE magazine</a></li>
    <li><a tabindex="-1" href="#">WordPress DevelopmentS</a></li>
    <li><a tabindex="-1" href="#">Speaking Engagements</a></li>
    <li class="divider"></li>
    <li><a tabindex="-1" href="#">Social Media</a></li>
  </ul>
</li>
```

If you need to keep links intact (which is useful if the browser is not enabling JavaScript), use the `data-target` attribute along with `href="#"`:

```
<div class="dropdown">
  <a class="dropdown-toggle" id="dLabel" role="button"
  data-toggle="dropdown" data-target="#" href="/page.html">
    Dropdown
    <b class="caret"></b>
```

```
    </a>
    <ul class="dropdown-menu" role="menu" aria-labelledby="dLabel">
      ...
    </ul>
  </div>
```

Dropdown Usage via JavaScript

To call the dropdown toggle via JavaScript, use the following method:

```
$('.dropdown-toggle').dropdown()
```

Method

The dropdown toggle has a simple method to show or hide the dropdown. There are no options:

```
$().dropdown('toggle')
```

Scrollspy

The Scrollspy plugin (Figure 4-3) allows you to target sections of the page based on scroll position. In its basic implementation, as you scroll, you can add `.active` classes to the navbar based on the scroll position. To add the Scrollspy plugin via data attributes, add `data-spy="scroll"` to the element you want to spy on (typically the body) and `data-target=".navbar"` to the navbar that you want to apply the class changes to. For this to work, you must have elements in the body of the page that have matching IDs of the links that you are spying on.

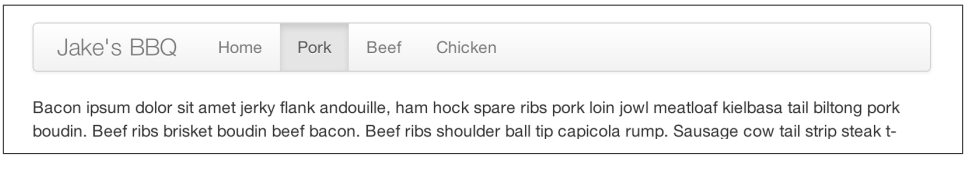

Figure 4-3. Scrollspy example

Usage

For Scrollspy, you will need to add `data-spy="scroll"` to the `<body>` tag, along with `data-target=".navbar"` that references the element that you are spying on:

```
<body data-spy="scroll" data-target=".navbar">...</body>
```

In the navbar, you will need to have page anchors that will serve as indicators for the element to spy on:

```
<div class="navbar">
  <div class="navbar-inner">
    <div class="container">
      <a class="brand" href="#">Jake's BBQ</a>
      <div class="nav-collapse">
        <ul class="nav">
          <li class="active"><a href="#">Home</a></li>
          <li><a href="#pork">Pork</a></li>
          <li><a href="#beef">Beef</a></li>
          <li><a href="#chicken">Chicken</a></li>
        </ul>
      </div><!-- /.nav-collapse -->
    </div>
  </div><!-- /navbar-inner -->
</div>
```

Usage via JavaScript

If you would rather invoke the scrollspy with JavaScript instead of using the data attributes, you can do so by selecting the element to spy on, and then invoking the `.scrollspy()` function:

```
$('#navbar').scrollspy()
```

.scrollspy('refresh') Method

When calling the scrollspy via the JavaScript method, you need to call the `.refresh` method to update the DOM. This is helpful if any elements of the DOM have changed.

```
$('[data-spy="scroll"]').each(function () {
        var $spy = $(this).scrollspy('refresh')
});
```

Options

Options can be passed via data attributes or JavaScript. For data attributes, prepend the option name to `data-`, as in `data-offset=""` (see Table 4-3).

Table 4-3. Scrollspy option

Name	Type	Default	Description
offset	number	10	Pixels to offset from top of page when calculating position of scroll.

The offset option is handy when you are using a fixed navbar. You will want to offset the scroll by about 50 pixels so that it reads at the correct time (see Table 4-4).

Event

Table 4-4. Scrollspy event

Event	Description
activate	This event fires whenever a new item becomes activated by the scrollspy.

Toggleable Tabs

Tabbable tabs were introduced in Chapter 3. By combining a few data attributes, you can easily create a tabbed interface (Figure 4-4). To do so, create the nav interface, and then wrap the content of the tabs inside a `<div>` with a class of `.tab-content`:

```
<ul class="nav nav-tabs">
    <li><a href="#home" data-toggle="tab">Home</a></li>
        <li><a href="#profile" data-toggle="tab">Profile</a></li>
        <li><a href="#messages" data-toggle="tab">Messages</a></li>
        <li><a href="#settings" data-toggle="tab">Settings</a></li>
</ul>

<div class="tab-content">
        <div class="tab-pane active" id="home">...</div>
        <div class="tab-pane" id="profile">...</div>
        <div class="tab-pane" id="messages">...</div>
        <div class="tab-pane" id="settings">...</div>
</div>
```

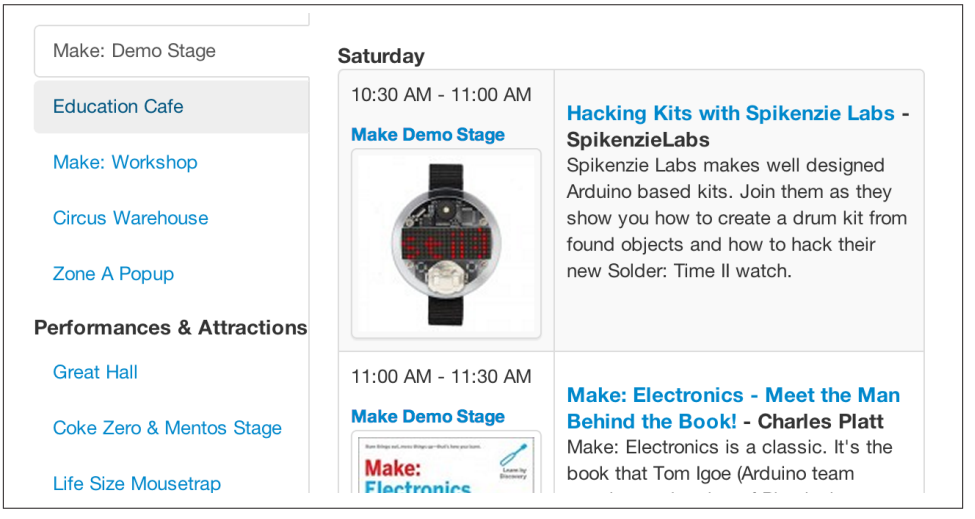

Figure 4-4. Toggleable tabs

Usage

To enable the tabs, you can use the Bootstrap Data API or use JavaScript directly. With the Data API, you need to add `data-toggle` to the anchors. The anchor targets will activate the element that has the `.tab-pane` class and relative ID. Alternatively, `data-target=""` may be used instead of `href="#"` to apply the same action. Here is one way to enable tabs:

```
 $('#myTab a').click(function (e) {
        e.preventDefault();
        $(this).tab('show');
})
```

Here's an example of different ways to activate tabs:

```
$('#myTab a[href="#profile"]').tab('show'); // Select tab by name
$('#myTab a:first').tab('show'); // Select first tab
$('#myTab a:last').tab('show'); // Select last tab
$('#myTab li:eq(2) a').tab('show'); // Select third tab (0-indexed)
```

Events

Tabs panes have two different events that can be hooked into, as shown in Table 4-5.

Table 4-5. Toggleable tab events

Event	Description
show	This event fires on tab show, but before the new tab has been shown. Use `event.target` and `event.relatedTarget` to target the active tab and the previous active tab (if available), respectively.
shown	This event fires on tab show after a tab has been shown. Use `event.target` and `event.relatedTarget` to target the active tab and the previous active tab (if available), respectively.

Here's a code example of a shown method:

```
$('a[data-toggle="tab"]').on('shown', function (e) {
        e.target // activated tab
        e.relatedTarget // previous tab
})
```

For information about the `.on` method, refer to the jQuery website.

Tooltips

Tooltips (Figure 4-5) are useful when you need to describe a link or (used in conjunction with the `<abbr>` tag) provide the definition of an abbreviation. The plugin was originally based on the *jQuery.tipsy* plugin written by Jason Frame. Tooltips have since been updated to work without images, animate with a CSS animation, and work with the Bootstrap JavaScript API.

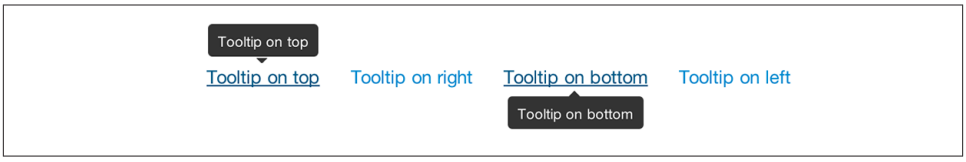

Figure 4-5. Tooltip placement

Usage

To add a tooltip, add `rel="tooltip"` to an anchor tag. The title of the anchor will be the text of a tooltip. The following two examples show how to do this in the Bootstrap Data API and JavaScript, respectively:

```
<a href="#" rel="tooltip" title="This is the tooltip">Tooltip Example</a>
```

```
$('#example').tooltip(options)
```

Options

Like all of the plugins, there are options that can be added via the Bootstrap Data API or invoked via JavaScript. All options need to have `data-` prepended to them. So, the `title` option would become `data-title` (see Table 4-6).

Table 4-6. Tooltip options

Name	Type	Default	Description
animation	Boolean	true	Applies a CSS fade transition to the tooltip.
html	Boolean	false	Inserts HTML into the tooltip. If false, jQuery's `text` method will be used to insert content into the dom. Use text if you're worried about XSS attacks.
placement	string/function	'top'	Specifies how to position the tooltip (i.e., top, bottom, left, or right).
selector	string	false	If a selector is provided, tooltip objects will be delegated to the specified targets.
title	string/function	''	The title option is the default title value if the `title` attribute isn't present.
trigger	string	'hover'	Defines how the tooltip is triggered: click, hover, focus, or manually.
delay	number/object	0	Delays showing and hiding the tooltip in ms—does not apply to manual trigger type. If a number is supplied, delay is applied to both hide/show. Object structure is: `delay: { show: 500, hide: 100 }`

Methods

Here are some useful methods for tooltips.

Options

Attaches a tooltip handler to an element collection:

```
$().tooltip(options)
```

Show

Reveals an element's tooltip:

```
$('#element').tooltip('show')
```

Hide

Hides an element's tooltip:

```
$('#element').tooltip('hide')
```

Toggle

Toggles an element's tooltip:

```
$('#element').tooltip('toggle')
```

Destroy

Hides and destroys an element's tooltip:

```
$('#element').tooltip('destroy')
```

Popover

The popover (see Figure 4-6) is a sibling of the tooltip, offering an extended view complete with a heading. For the popover to activate, a user just needs to hover the cursor over the element. The content of the popover can be populated entirely using the Bootstrap Data API. This method requires a tooltip.

Figure 4-6. Popover placement

Use the following code for popover placement:

```
<a href="#" class="btn" rel="popover" title="Using Popover"
data-content="Just add content to the data-content attribute.">Click Me!</a>
```

Usage

To enable the popover with JavaScript, use the `.popover()` function, passing in any options that you might need:

```
$('#example').popover(options)
```

Options

All options can be passed via the Bootstrap Data API, or directly with JavaScript (see Table 4-7).

Table 4-7. Popover options

Name	Type	Default	Description
animation	Boolean	true	Applies a CSS fade transition to the tooltip.
html	Boolean	false	Inserts HTML into the popover. If false, jQuery's `text` method will be used to insert content into the dom. Use text if you're worried about XSS attacks.
placement	string	function	*right*
Specifies how to position the popover (i.e., top, bottom, left, right)	selector	string	false
If a selector is provided, tooltip objects will be delegated to the specified targets.	trigger	string	*click*
How the popover is triggered (i.e., click, hover, focus, manual)	title	string	function
''	Default title value if *title* attribute isn't present	content	string
function	''	Default content value if *data-content* attribute isn't present	delay
number	object	0	Delays showing and hiding the popover in ms—does not apply to manual trigger type. If a number is supplied, delay is applied to both hide/show. Object structure is: `delay: {show: 500, hide: 100 }`.

Methods

Here are some useful methods for popovers.

Options

Initializes popovers for an element collection:

```
$().popover(options)
```

Show

Reveals an element's popover:

```
$('#element').popover('show')
```

Hide

Hides an element's popover:

```
$('#element').popover('hide')
```

Toggle

Toggles an element's popover:

```
$('#element').popover('toggle')
```

Destroy

Hides and destroys an element's popover:

```
$('#element').popover('destroy')
```

Alerts

With the Data API, it is easy to add dismiss functionality to alert messages (Figure 4-7).

Figure 4-7. Error alert message

Usage

You can close an alert manually with the JavaScript `.alert()` method or use data attributes in conjunction with an anchor or button.

Here is how to dismiss via JavaScript:

```
$(".alert").alert()
```

Here is how to dismiss via Data API:

```
<a class="close" data-dismiss="alert" href="#">&times;</a>
```

Close Method

To enable all alerts to be closed, add the following method. To enable alerts to animate out when closed, make sure they have the `.fade` and `.in` class already applied to them.

```
$(".alert").alert('close')
```

Events

There are two events that can be tied to Bootstrap's `alert` class as shown in Table 4-8.

Table 4-8. Alert class events

Event	Description
close	This event fires immediately when the close instance method is called.
closed	This event is fired when the alert has been closed (will wait for CSS transitions to complete).

As an example, if you wanted to trigger a function after an alert has closed, you could use this function:

```
$('#my-alert').bind('closed', function () {
  // do something…
})
```

Buttons

Buttons were introduced in Chapter 3. With Bootstrap, you don't need to do anything to make them work as links or as buttons in forms. With this plugin you can add in some interaction, such as loading states or button groups with toolbar-like functionality.

Loading State

To add a loading state to a button (shown in Figure 4-8), simply add `data-loading-text="Loading..."` as an attribute to the button:

```
<button type="button" class="btn btn-primary" data-loading-text="Loading...">
Submit!</button>
```

When the button is clicked, the `.disabled` class is added, giving the appearance that it can no longer be clicked.

Figure 4-8. Loading button

Single Toggle

When clicking on a button with the `data-toggle="button"` attribute (Figure 4-9), a class of `.active` is added:

```
<button type="button" class="btn btn-primary" data-toggle="button">Toggle
</button>
```

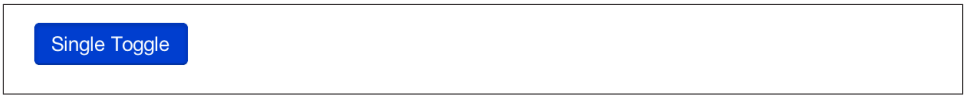

Figure 4-9. Toggle button

Checkbox Buttons

Buttons can work like checkboxes (as in Figure 4-10), allowing a user to select many of the options in a button group. To add this function, add `data-toggle="buttons-checkbox"` for checkbox style toggling on `.btn-group`:

```
<div class="btn-group" data-toggle="buttons-checkbox">
  <button type="button" class="btn btn-primary">Left</button>
  <button type="button" class="btn btn-primary">Middle</button>
  <button type="button" class="btn btn-primary">Right</button>
</div>
```

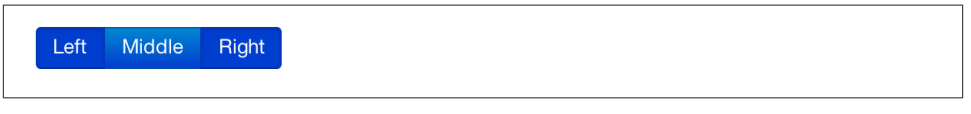

Figure 4-10. Checkbox buttons

Radio Buttons

Radio buttons (Figure 4-11) function similarly to checkboxes. The primary difference is that a radio button doesn't allow for multiple selections—only one button in the group can be selected. To add radio-style toggling on `btn-group`, add `data-toggle="buttons-radio"`:

```
<div class="btn-group" data-toggle="buttons-radio">
  <button type="button" class="btn btn-primary">Left</button>
  <button type="button" class="btn btn-primary">Middle</button>
  <button type="button" class="btn btn-primary">Right</button>
</div>
```

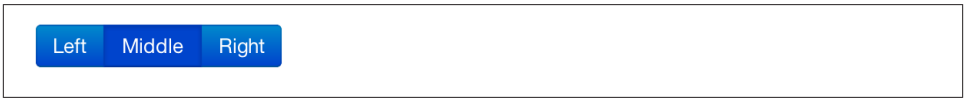

Figure 4-11. Radio buttons

Usage

The `.button` method can be applied to any class or ID. To enable all buttons in the `.nav-tabs` via JavaScript, add the following code:

```
$('.nav-tabs').button()
```

Methods

The following methods are useful to use with buttons.

Toggle

Toggles push state. Gives the button the appearance that it has been activated:

```
$().button('toggle')
```

Loading

When loading, the button is disabled and the text is changed to the option from the `data-loading-text` attribute:

```
<button type="button" class="btn" data-loading-text="loading stuff..." >...
</button>
```

Reset

Resets button state, bringing the original content back to the text. This method is useful when you need to return the button back to the primary state:

```
$().button('reset')
```

String

String in this method is referring to any string declared by the user:

```
$().button('string')
```

To reset the button state and bring in new content, use the string method:

```
<button type="button" class="btn" data-complete-text="finished!" >...</button>

<script>
  $('.btn').button('complete')
</script>
```

Collapse

The collapse plugin makes it easy to make collapsing divisions of the page (see Figure 4-12). Whether you use it to build accordion navigation or content boxes, it allows for a lot of content options.

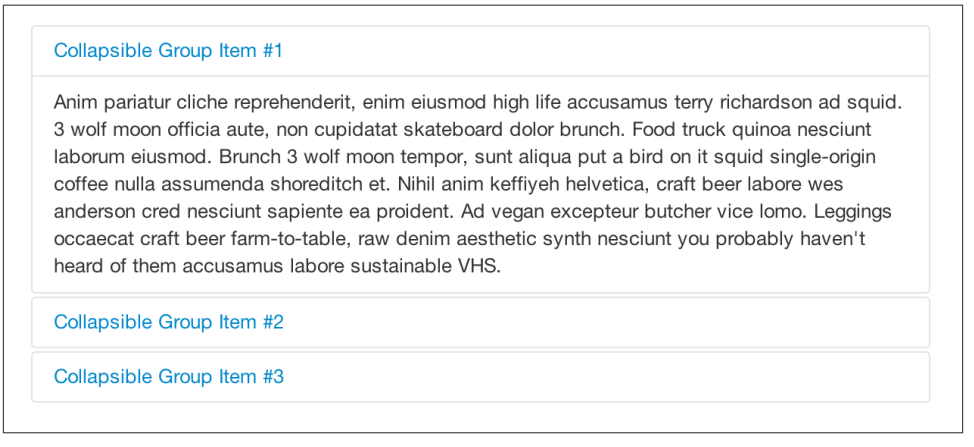

Figure 4-12. Accordion

The following code creates collapsible groups:

```
<div class="accordion" id="accordion2">
  <div class="accordion-group">
    <div class="accordion-heading">
      <a class="accordion-toggle" data-toggle="collapse" data-parent="#accordion2"
      href="#collapseOne">
        Collapsible Group Item #1
      </a>
    </div>
    <div id="collapseOne" class="accordion-body collapse in">
      <div class="accordion-inner">
        Anim pariatur cliche...
```

```
      </div>
    </div>
  </div>
  <div class="accordion-group">
    <div class="accordion-heading">
      <a class="accordion-toggle" data-toggle="collapse" data-parent="#accordion2"
      href="#collapseTwo">
        Collapsible Group Item #2
      </a>
    </div>
    <div id="collapseTwo" class="accordion-body collapse">
      <div class="accordion-inner">
        Anim pariatur cliche...
      </div>
    </div>
  </div>
</div>
...
```

You can also use the data attributes to make all content collapsible:

```
<button type="button" class="btn btn-danger" data-toggle="collapse"
data-target="#demo">
  simple collapsible
</button>

<div id="demo" class="collapse in"> … </div>
```

Usage

Via data attributes

Like all of the plugins that use the Data API, you can add all needed markup without writing any JavaScript. Add `data-toggle="collapse"` and a `data-target` to the element to automatically assign control of a collapsible element. The `data-target` attribute will accept a CSS selector to apply the collapse to. Be sure to add the class `.collapse` to the collapsible element. If you'd like it to default open, include the additional class `.in`.

To add accordion-like group management to a collapsible control, add the data attribute `data-parent="#selector"`.

Via JavaScript

The collapse method can activated with JavaScript as well:

```
$(".collapse").collapse()
```

Options

The options listed in Table 4-9 can be passed via data attributes or with JavaScript.

Table 4-9. Collapse options

Name	Type	Default	Description
parent	selector	false	If selector, then all collapsible elements under the specified parent will be closed when this collapsible item is shown. (Similar to traditional accordion behavior.)
toggle	Boolean	true	Toggles the collapsible element on invocation.

Methods

The following methods are useful to use with collapsible elements.

Options

Activates your content as a collapsible element. Accepts an optional options object:

```
.collapse(options)
```

Toggle

Toggles a collapsible element to shown or hidden:

```
$('#myCollapsible').collapse({
  toggle: false
})
.collapse('toggle')
```

Show

Shows a collapsible element:

```
.collapse('show')
```

Hide

Hides a collapsible element:

```
.collapse('hide')
```

Events

There are four events that can be hooked into with the collapse plugin, described in Table 4-10.

Table 4-10. Collapse events

Event	Description
show	This event fires immediately when the `show` instance method is called.
shown	This event is fired when a collapse element has been made visible to the user (will wait for CSS transitions to complete).
hide	This event is fired immediately when the `hide` method has been called.
hidden	This event is fired when a collapse element has been hidden from the user (will wait for CSS transitions to complete).

After a has been collapsed, you could use the following code to execute a function:

```
$('#myCollapsible').on('hidden', function () {
  // do something…
})
```

Carousel

The Bootstrap carousel (Figure 4-13) is a flexible, responsive way to add a slider to your site. In addition to being responsive, the content is flexible enough to allow images, iframes, videos, or just about any type of content that you might want.

Figure 4-13. Carousel

The following code creates the Bootstrap carousel:

```
<div id="myCarousel" class="carousel slide">
  <!-- Carousel items -->
  <div class="carousel-inner">
    <div class="active item">…</div>
    <div class="item">…</div>
    <div class="item">…</div>
  </div>
  <!-- Carousel nav -->
  <a class="carousel-control left" href="#myCarousel" data-slide="prev">&lsaquo;</a>
  <a class="carousel-control right" href="#myCarousel" data-slide="next">&rsaquo;</a>
</div>
```

Usage

To implement the carousel, you just need to add the code with the markup above. There is no need for data attributes, just simple class-based development. You can manually call the carousel with JavaScript, using the following code:

```
$('.carousel').carousel()
```

Options

Options can be passed through data attributes or through JavaScript. The options are listed in Table 4-11.

Table 4-11. Carousel options

Name	Type	Default	Description
interval	number	5000	The amount of time to delay between automatically cycling an item. If false, carousel will not automatically cycle.
pause	string	"hover"	Pauses the cycling of the carousel on mouseenter and resumes the cycling of the carousel on mouseleave.

Methods

The following methods are useful carousel code.

Options

Initializes the carousel with an optional `options` object and starts cycling through items:

```
$('.carousel').carousel({
  interval: 2000
})
```

Cycle

Cycles through the carousel items from left to right:

```
.carousel('cycle')
```

Pause

Stops the carousel from cycling through items:

```
.carousel('pause')
```

Number

Cycles the carousel to a particular frame (0-based, similar to an array):

```
.carousel('number')
```

Prev

Cycles to the previous item:

```
.carousel('prev')
```

Next

Cycles to the next item:

```
.carousel('next')
```

Events

The carousel has two events that can be hooked into, described in Table 4-12:

Table 4-12. Carousel events

Event	Description
slide	This event fires immediately when the slide instance method is invoked.
slid	This event is fired when the carousel has completed its slide transition.

Typeahead

Typeahead allows you to easily create typeahead inputs in forms (Figure 4-14). For example, you could preload states in a state field or, with some JavaScript, get search results using some AJAX calls.

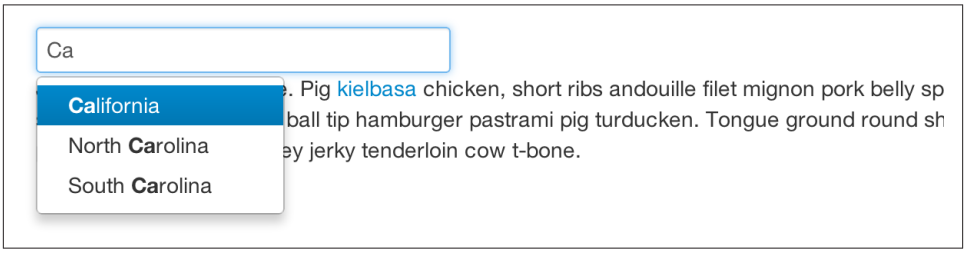

Figure 4-14. Typeahead

Usage

Using Data API, you can add sources via the `data-source` attribute. Items should be listed in either a JSON array or a function:

```
<input
        type="text"
        class="span3"
        data-provide="typeahead"
        data-items="4"
```

```
        data-source="[
                'Alabama',
                'Alaska',
                'Arizona',
                'Arkansas',
                'California',
                ...
                ]"
>
```

To call directly with JavaScript, use the following method:

```
$('.typeahead').typeahead()
```

Options

Table 4-13 shows a list of options.

Table 4-13. Carousel options

Name	Type	Default	Description
source	array, function	[]	The data source to query against. May be an array of strings or a function. The function is passed through two arguments: the `query` value in the input field and the `process` callback. The function may be used synchronously by returning the data source directly or asynchronously via the `process` callback's single argument.
items	number	8	The maximum number of items to display in the dropdown.
minLength	number	1	The minimum character length needed before triggering autocomplete suggestions.
matcher	function	case insensitive	The method used to determine if a query matches an item. Accepts a single argument, the item against which to test the query. Accesses the current query with `this.query`. Return a Boolean true if query is a match.
sorter	function	exact match, case sensitive, case insensitive	Method used to sort autocomplete results. Accepts a single argument item and has the scope of the typeahead instance. Reference the current query with `this.query`.
updater	function	returns selected item	The method used to return the selected item. Accepts a single argument item and has the scope of the typeahead instance.
highlighter	function	highlights all default matches	Method used to highlight autocomplete results. Accepts a single argument item and has the scope of the typeahead instance. Should return HTML.

Affix

The affix plugin allows a `<div>` to become affixed to a location on the page. A common example of this is social icons. They will start in a location, but as the page hits a certain mark, the `<div>` will become locked in place and will stop scrolling with the rest of the page.

Usage

To apply the affix plugin to a `<div>`, you can use either data attributes, or you can use JavaScript directly. Note that you must position the element so that it can be affixed to the page. Position is controlled by the `data-spy` attribute, using either `affix`, `affix-top`, or `affix-bottom`. You then use the `data-offset` to calculate the position of the scroll.

```
<div data-spy="affix" data-offset-top="200">
        ...
</div>
```

Option

Name	Type	Default	Description
offset	number/ function/ object	10	Pixels to offset from screen when calculating position of scroll. If a single number is provided, the offset will be applied in both top and left directions. To listen for a single direction or multiple unique offsets, just provide an object `offset: { x: 10 }`. Use a function when you need to dynamically provide an offset (useful for some responsive designs).